Route Consistency Vehicle Routing: a Bi-Objective Approach

Victor Pillac, Christelle Gueret, Andrés Medaglia

Route Stability in Dynamic Vehicle Routing: a Bi-Objective Approach

Victor Pillac[1,2], Christelle Guéret[1], Andrés L. Medaglia[2]

[1] LUNAM Université, École des Mines de Nantes, IRCCyN UMR 6597, Nantes, France
{vpillac,gueret}@mines-nantes.fr

[2] Universidad de los Andes, Industrial Engineering Department, Bogotá, Colombia
amedagli@uniandes.edu.co

Mots-clés : *recherche opérationnelle, optimisation, tournées de véhicules, tournées dynamiques, optimisation bi-objectif.*

1 Introduction

Vehicle Routing Problems (VRPs) consider the operation of a fleet of vehicles that need to service customer requests. The underlying problem consists in designing a set of routes that visit all customers, optimizing one or multiple objectives.

Dynamic Vehicle Routing Problems (D-VRPs) are an extension of classical VRPs in which the information available to the decision maker changes or is updated dynamically. The most common source of dynamism studied in the literature is the apparition of new customer requests, that need to be incorporated in the vehicles current routes. On top of this basic definition, authors have studied many variants and have proposed different optimization approaches [3].

To the best of our knowledge, all studies on D-VRPs focus on the optimization of a single criterion, such as the minimization of the total traveled distance, or the maximization of the number of served customers. On the other hand, a growing number of studies consider multiple objectives for static VRPs in an attempt to better model the reality of routing application, as surveyed by Jozefowiez et al. [1].

Most studies on D-VRPs consider that routes can be designed online, which means that vehicle drivers do not know their next destination until they finish serving their current customer. Although this assumption is theoretically appealing, it may not be desirable in a practical context in which drivers are used to know their routes from the beginning of the day.

In this work we propose an optimization algorithm able to optimize the minimization of a cost function (the total traveled distance), and the minimization of the changes made in the vehicles routes in a dynamic routing context, and we study the tradeoff between both objectives.

2 Problem definition

We focus on the D-VRP with time windows (D-VRPTW), in which a limited fleet of identical capacitated vehicles must deliver a unique product to a set of customers over a single day horizon. Each customer has a geographic position and requires a known quantity of product, and must be serviced within a given time frame. While a set of (*static*) customers is known beforehand, new (*dynamic*) customers may appear during the day.

The problem is first to design an initial set of routes, visiting all the static customers. Then each time a new customer appear, to decide whether it can be served or not, and eventually reoptimize the vehicle routes to include it considering two objectives: the minimization of the traveled distance, and the minimization of the number of changes made to the routes. We assume that rejecting customers induce a penalty that can be interpreted as an outsourcing cost.

We use the Levenshtein (or edit) distance to measure the number of changes made to the routes after a reoptimization. Levenshtein distance between two routes is defined by the minimum number of insertions, removals, or substitutions of customers that have to be applied to transform the reference route into the new route. FIG. 1 illustrates a case in which the distance between the reference and new route is 3: 1 substitution (SUB), 1 insertion (INS), and 1 removal (REM).

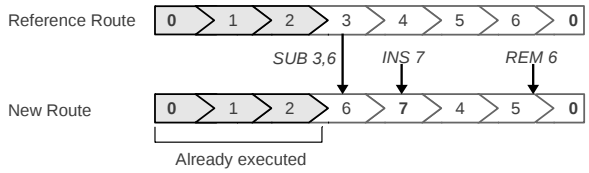


FIG. 1: Example of the Levenshtein distance between two routes.

3 The proposed approach

The proposed approach, namely parallel Bi-objective Adaptive Large Neighborhood Search (pBiALNS), is an extension of the Adaptive Large Neighborhood Search (ALNS) algorithm [4], itself inspired by the Large Neighborhood Search (LNS) algorithm [6]. LNS works by successively destroying (removing customers) and repairing (inserting customers back) a current solution, using destroy and repair *operators*. ALNS adds an adaptive layer that randomly selects operators depending on their past performance. This allows an auto-fitting of the algorithm to the problem at hand.

An adaptation of ALNS to a bi-objective problem was proposed by Schmid and Hartl [5], and we extended it to take advantage of parallel computing architectures. In a nutshell, pBiALNS maintains and optimizes a set of non-dominated and possibly infeasible solutions. ALG. 1 presents the outline of our implementation.

ALG. 1 Parallel Bi-objective Adaptive Large Neighborhood Search (pBiALNS) algorithm

Input: Π_0 an initial solution, n the number of parallel iterations, T the number of parallel threads, N the number of sequential iterations.

Output: The non-dominated solutions found by the algorithm.

```
Initialize the non-dominated solutions: Ω ← {Π_0}
for N iterations do
    Select subset Ω_t of T non-dominated solutions
    parallel forall Π in Ω_t do
        In a separate thread, perform n ALNS iterations starting with Π
        Report the non-dominated solutions to the main thread
    end forall
    Update the non-dominated solutions Ω
return The non-dominated solutions Ω
```

Note that pBiALNS allows infeasible solutions that do not visit all customers. Therefore, we define a dominance relation that ensures that no feasible solution will be dominated by an infeasible solution:

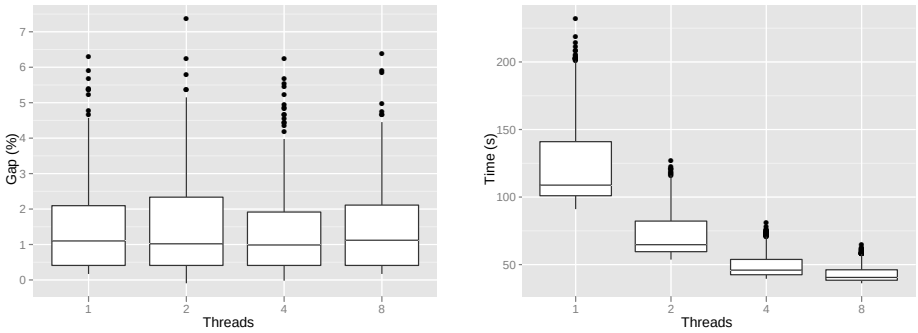
Definition 1 (Dominance) *A solution* Π *dominates (denoted* $\prec$*) a solution* Π' *if and only if* Π *is as good as* Π' *on both objectives, and strictly better on one objective, and either* Π *is feasible or both* Π *and* Π' *are infeasible.*

4 Computational results

To assess the effect of parallelization we tested our algorithm on the static single objective instances proposed by Solomon [7] on a quad-core desktop computer[1]. To this purpose we

[1]CPU: Intel i7 860 (4x2.8GHz), RAM: 6GB DDR3, Ubuntu 11.04 x64, Java 7, $N.T.n = 25000$ iterations

slightly modified pBiALNS and only considered a fixed size set of solutions for Ω, leading to algorithm pALNS. As illustrated by FIG 2, increasing the number of threads has no significant impact on the gap to the best known solutions which is of around 1%, but it allows a reduction of running times by a factor 3.



(a) Gap vs. number of threads (b) Gap vs. CPU Time

FIG. 2: Influence of the number of threads on CPU time and GAP.

We tested the pBiALNS algorithm on a subset of 72 D-VRPTW 100 customer instances proposed by Lackner [2] and based on the Solomon [7] benchmark, in which a proportion δ of the customers is revealed dynamically. Therefore, we perform a simulation in which we run the pBiALNS algorithm for 5000 iterations each time a new request is released to produce a solution that will be used until the next customer is revealed. The simulation starts with an *initial solution* containing all the requests known initially, and leads to a *final solution*.

FIG. 3 represents the objective space explored by pBiALNS at a given time of the simulation for one instance. The chart illustrates the variety of solutions to choose from at each decision, ranging from the one best in terms of cost (upper left corner) to the one closest to the current solution (lower right corner). Therefore we define a *selection policy* to select one solution from the set of non-dominated: we select the solution (green diamond) that is closest to the reference, allowing a deviation of at most γ from the best solution in terms of cost.

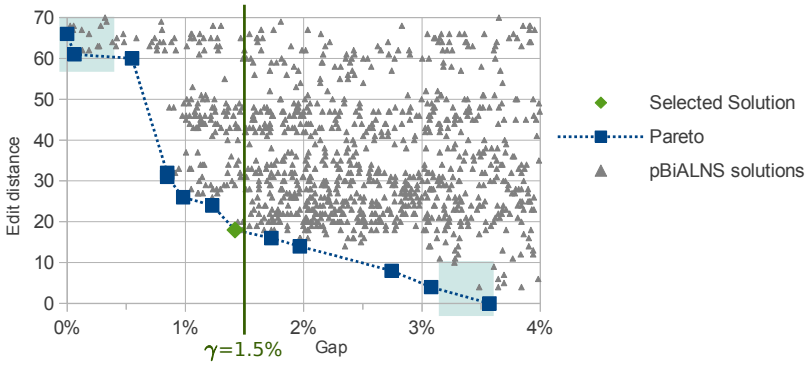


FIG. 3: Objective space for instance R101

TAB. 1 presents (a) the average edit distance between the final solution and the initial solution, (b) and the gap between the cost of the final solution and the cost of a solution evaluated a-posteriori, for different values of γ and degrees of dynamism (δ). Running times are of 11 seconds on average at each decision. As expected the distance is negatively correlated to γ, and is minimal for $\gamma = \infty$. In this case we always choose the solution which is the closest to the reference solution, in other words we insert new requests at their best position, which leads to a distance equal to the number of inserted requests. It is important to note that the quality of the routing, measured by the gap to the static solution, is positively correlated to γ. This confirms the intuition that poor routing decisions tend to add up over time and can lead to great deviations at the end of the day. Our results also indicate that, for problems with low

degree of dynamism, it can be worth sacrificing quality of solution to gain in route stability (see for instance $\delta = 10, \gamma = 5\%$). However, this statement no longer holds for instances with higher degrees of dynamism where numerous changes are necessary to insert all requests. In this case it is better to focus on optimizing the routing, as it does not lead to excessive instability in routes.

(a) Distance to initial solution

	γ				
δ	**0%**	**1%**	**2%**	**5%**	∞
10	30.5	20.6	16.0	13.5	10.0
50	74.5	72.5	69.5	65.7	50.0
90	96.6	96.4	96.3	96.2	90.0

(b) Gap to a-posteriori solution

	γ				
δ	**0%**	**1%**	**2%**	**5%**	∞
10	1.95%	2.32%	2.53%	2.55%	12.66%
50	6.04%	6.48%	7.36%	10.52%	44.18%
90	11.32%	11.64%	14.77%	20.08%	91.98%

TAB. 1: Preliminary computational results (each figure is an average value over 24 instances)

5 Conclusions

In this study, we proposed a fast optimization approach for a bi-objective and dynamic vehicle routing problem (namely pBiALNS). We showed that the parallelization of the algorithm brings factor 3 speedups on a regular desktop machine, allowing its use in a dynamic context where routing decisions have to be made in limited time.

In addition we investigated the tradeoff between the stability of routes and the quality of routing. We showed that the objectives are contradictory in nature. This suggests that real-world applications should not ignore the practical implications of designing vehicle routes in an online fashion, as it induces numerous changes in the driver's schedule, but also that naive approaches ultimately result in very poor routing.

Acknowledgements Financial support for this work was provided by the CPER (Contrat de Projet Etat Region) Vallée du Libre (France); and the Centro de Estudios Interdisciplinarios Básicos y Aplicados en Complejidad (CEIBA, Colombia). This support is gratefully acknowledged.

References

[1] Jozefowiez, N., Semet, F., and Talbi, E.-G. (2008). Multi-objective vehicle routing problems. *European Journal of Operational Research*, 189(2):293 – 309.

[2] Lackner, A. (2004). *Dynamische Tourenplanung mit ausgewählten Metaheuristiken*, volume 47 of *Göttinger Wirtschaftsinformatik*. Cuvillier.

[3] Pillac, V., Gendreau, M., Guéret, C., and Medaglia, A. L. (2011). A review of dynamic vehicle routing problems. Technical report, CIRRELT. CIRRELT-2011-62.

[4] Pisinger, D. and Ropke, S. (2007). A general heuristic for vehicle routing problems. *Computers & Operations Research*, 34(8):2403–2435.

[5] Schmid, V. and Hartl, R. F. (2011). Large neighborhood search for solving the Bi-Objective Capacitated m-Ring-Star Problem. In Di Gaspero, L., Schaerf, A., and Stützle, T., editors, *Proceedings of the 9th Metaheuristics Conference (MIC 2011)*, pages 700–703. Università degli Studi di Udine.

[6] Shaw, P. (1998). Using constraint programming and local search methods to solve vehicle routing problems. In *Principles and Practice of Constraint Programming – CP98*, volume 1520 of *Lecture Notes in Computer Science*, pages 417–431. Springer Berlin / Heidelberg.

[7] Solomon, M. M. (1987). Algorithms for the vehicle-routing and scheduling problems with time window constraints. *Operations Research*, 35(2):254–265.